Un schéma (abstrait) d'itération répartie. Application au calcul des chemins de valeurs minimales

Jean-Michel Hélary, Michel Raynal

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 879

UN SCHEMA (ABSTRAIT) D'ITERATION REPARTIE. APPLICATION AU CALCUL DES CHEMINS DE VALEURS MINIMALES

Jean-Michel HELARY
Michel RAYNAL

JUILLET 1988

RR-0879

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042-RENNES CÉDEX
FRANCE
Téléphone: 99 36 20 00
Télex: UNIRISA 950 473 F
Télécopie: 99 38 38 32

Publication Interne n° 417
24 Pages - Juin 1988

Un schéma (abstrait) d'itération répartie. Application au calcul des chemins de valeurs minimales.

An abstract distributed iteration scheme. Application to the computation of weighted shortest paths

Jean-Michel HELARY, Michel RAYNAL
IRISA-IFSIC – Campus de Beaulieu– 35042 RENNES CEDEX
E.mail: helary@irisa.fr,raynal@irisa.fr

Résumé

La définition de structures de contrôle aptes à exprimer les algorithmes distribués est un problème central de l'algorithmique répartie. On examine ici l'une de ces structures: l'itération répartie. Après en avoir distingué deux formes aux propriétés différentes, appelées respectivement train de vagues et séquence de phases, on s'intéresse plus particulièrement au schéma abstrait que constitue la vague, sur lequel est bâtie la première des deux formes d'itération répartie. Ce schéma est défini à la manière d'un type abstrait, c'est-à-dire par les opérations de contrôle qu'il offre à l'utilisateur et les propriétés qui leur sont associées, indépendamment d'une mise en œuvre particulière. Deux exemples de mise en œuvre sont ensuite proposés; elles s'appuient respectivement sur les topologies de contrôle que sont l'anneau virtuel et l'arborescence couvrante. On montre ensuite comment un tel schéma d'itération répartie peut être exploité pour la construction méthodique d'algorithmes distribués; l'exemple choisi est relatif au calcul de chemins de valeurs minimales. Outre la définition d'un nouvel algorithme distribué performant, la méthode proposée permet d'illustrer, dans le contexte réparti, une technique d'utilisation systématique de l'itération.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

Abstract

Defining control structures able to express distributed algorithms is a central problem in distributed algorithmics. One of these structures is inspected: distributed iteration. Two kinds of distributed iteration with different properties are characterized, namely wave and phase sequences. The wave control scheme is a basis for the wave sequence; this scheme is defined in the same way as an abstract type, that is to say in terms of control operations available to user and their associated properties irrespective of any particular implementation. Two particular implementations, each one lying on a control topology (virtual ring and spanning directed tree) are sketched. Lastly, it is shown that the iteration control structure previously defined can be used for a methodical construction of distributed algorithms; the application case is related to the calculus of weighted shortest paths issued from a given vertex. Beyond obtaining a new and performant algorithm (with a $O(n^2)$ message complexity, where n is the number of vertices), the method is fitted to a systematic building technique of distributed iteration.

mots clefs: Algorithmique répartie, méthodologie, programmation structurée, type abstrait, structure de contrôle.

Table des matières

1 Introduction

La conception et la vérification des algorithmes répartis est une tâche difficile. Aussi un certain nombre d'auteurs [Seg83,Gaf86,HR88b] ont-ils proposé des méthodologies s'appuyant sur des techniques de conception modulaire. Parmi celles-ci la technique des blocs de construction (building blocks) proposée par Gafni semble particulièrement intéressante [Gaf86]; partant du fait que des classes d'algorithmes utilisent les mêmes structures de base, elle

propose un certain nombre de protocoles qui doivent constituer des "briques de base" ou des " squelettes" sur lesquels peuvent s'appuyer la conception et l'écriture d'un grand nombre d'algorithmes. La démarche n'est pas nouvelle: elle ne fait qu'étendre au contexte distribué les notions d'abstractions de données et de contrôle désormais classiques en programmation séquentielle [LZ74,WSL77] ou concurrente [Hoa74]. Il convient donc dans le domaine du calcul réparti d'isoler des abstractions fondamentales, de les étudier et de les maîtriser [Chan82,Che83,HR88c,Tel87,Tel88]. On se propose dans cet article l'étude de l'une des structures de contrôle de base d'un tel calcul: l'itération répartie. Après en avoir isolé deux formes le paragraphe 2 en présente une en détail sous la forme d'une structure de contrôle abstraite puis en propose plusieurs mises en œuvre.

La structure ainsi dégagée est ensuite utilisée au paragraphe 3 pour construire méthodiquement une version répartie de l'algorithme des chemins de valeur minimale issus d'un sommet donné, dans le cas de valeurs non négatives (dû à Dijkstra [Dij59]); par rapport aux algorithmes distribués connus, qui résolvent le même problème [CM82], celui-ci prend en compte l'hypothèse de non-négativité des valeurs des arcs.

2 Les schémas abstraits d'itération répartie

2.1 Le concept d'itération

Ce concept permet d'abstraire les notions de boucles et de calculs répétitifs. Une itération est composée de la description d'un calcul (règle de progression qui définit le pas de calcul de l'itération) et d'un critère d'arrêt, évalué à la fin de chaque pas. Toute itération est de plus caractérisée par un invariant qui décrit l'état des variables au début (et à la fin) de chacun des pas de l'itération. Il s'en suit que, dès lors que le critère d'arrêt est atteint, la conjonction de ce dernier et de l'invariant donne sa signification au résultat calculé.

2.2 Contexte réparti

On s'intéresse ici à une forme d'itération dans un contexte de parallèlisme et de répartition. Par parallèlisme on entend que l'algorithme est composé d'un certain nombre de processus (ou sites) coopérant à la réalisation d'un but commun. Par distribution on entend que cette coopération se fait uniquement par échange de messages (il n'y a donc pas de mémoire commune). Les

messages sont véhiculés par des canaux connectant des paires des processus; ces canaux sont supposés fiables et bi- directionnels; les délais de transit des messages sont arbitraires mais finis.

La structure globale d'un tel réseau de processus peut être modélisée par un graphe connexe non-orienté: les processus et les canaux en sont respectivement les sommets et les arêtes. Tout algorithme distribué particulier définit le réseau de processus nécessaire à la réalisation de son calcul.

2.3 Les formes d'itération répartie

Le calcul réalisé par le réseau de processus lors de chaque pas d'itération est distribué (et le nombre de processus qui y coopèrent peut différer d'un pas à l'autre). La décision de passer au pas suivant de l'itération (ou de terminer), requiert donc d'avoir détecté la terminaison du pas courant et d'avoir évalué la condition d'arrêt; il s'agit donc de récupérer les résultats et de détecter la terminaison de calculs locaux distribués dans les processus. Cette collecte et cette détection peuvent prendre diverses formes selon que le pas est contrôlé globalement par un seul processus ou individuellement par chaque processus; on parle d'algorithme à *vagues* dans le premier cas et d'algorithme à *phases* dans le second [HR88b].

2.4 Le concept de phase

Dans un algorithme à phases chacun des processus obéit au schéma suivant:

répéter

- *envoyer un message à chacun de ses voisins;*
- *recevoir un message de chacun de ses voisins;*
- *effectuer des calculs locaux*

jusqu'à *critère d'arrêt local*

Cette forme d'itération dont le pas est appelé phase a été proposée initialement par [Gal82] pour calculer des routages; elle a été étudiée par [BKR87] et [Kon87] qui, d'une part, ont caractérisé la classe des problèmes résolvables avec cette structure de contrôle et d'autre part, en ont exhibé un certain nombre de propriétés. Nous renvoyons le lecteur intéressé à ces références et à [HR88b] (chapitre 4) pour plus de détails sur cette structure

de contrôle répartie; remarquons toutefois la synchronisation qu'elle induit: si un processus exécute sa $p^{ième}$ phase ses voisins immédiats ne peuvent être qu'à la phase $p-1, p$ ou $p+1$; de manière générale 2 processus distants de q ont un décalage maximal de q phases à un instant donné.

2.5 Le concept de vagues

Dans cette forme d'itération, l'arrêt ou le passage au pas suivant est déterminé par un processus prédéfini que nous appelerons P_α. Lorsque P_α lance le pas suivant il est nécessaire qu'il en informe les autres processus (puisqu'ils participent a priori tous au calcul) et leur transmette les valeurs dont ils ont besoin afin d'exécuter leur part de calcul. De plus lorsque les processus ont terminé leurs calculs locaux il est nécessaire qu'ils en informent P_α et lui transmettent les résultats nécessaires à l'évaluation du critère d'arrêt. Ce schéma n'est rien d'autre que ce qu'il convenu d'appeler un parcours de réseau avec feedback; parcours de réseau car à partir de P_α un flot de contrôle parvient à chaque processus, avec feedback car ce flot retourne ensuite à P_α. Un tel flot de contrôle est appelé une *vague* [Sch85], et une succession séquentielle de tels flots un *train de vagues*.

Désignons par *vague(diffusé,collecté)* le flot de contrôle réalisant le parcours et véhiculant les valeurs *diffusé* et *collecté* qui représentent respectivement les données envoyées par P_α aux processus et les résultats renvoyés par ces derniers à P_α après le passage de la vague et leur calcul local.

Le comportement d'un train de vagues peut être facilement décrit par 4 primitives: *lancer*, *retour* utilisées par P_α et *visite*, *faire-suivre* utilisées par chaque autre processus.

Désignons par x_i^p l'évènement correspondant à la fin de la $p^{ième}$ exécution de la primitive x par le processus i, et par $\rightarrow$ la relation de précédence temporelle.

La séquentialité entre les vagues est alors spécifiée par:

$$(S) : \forall p > 0 : retour_\alpha^p \rightarrow lancer_\alpha^{p+1}$$

et les propriétés du flot de contrôle lors de la vague $n^o p$ s'expriment par:

$$(PF) : \forall p > 0, \forall j \neq \alpha : lancer_\alpha^p \rightarrow visite_j^p \rightarrow fairesuivre_j^p \rightarrow retour_\alpha^p$$

Le comportement temporel des primitives ainsi décrit, il est facile de donner les schémas de contrôle permettant aux processus de réaliser une itération contrôlée par P_α.

Comportement de P_α:

tant que <critère non vérifié>
faire
diffusé:= calcul (variables locales, collecté);
lancer *vague* (*diffusé*, $\emptyset$);
retour *vague(diffusé,collecté)*
fait

Remarque : le critère d'arrêt évalué par P_α est a priori global c'est à dire peut porter sur des valeurs issues de chacun des processus et amenées par la vague précédente.

Comportement de $Pi(i \neq \alpha)$

lors de visite *vague(diffusé,collecté)*
faire
<calculs locaux>;
r_i:= <résultat apporté par P_i à la vague présente >;
faire-suivre *vague* (*diffusé,collecté* $\cup r_i$)
fait

Un tel schéma itératif engendre une synchronsiation plus forte que celle issue des phases: lorsqu'un processus P_i est visité par la $p^{ième}$ vague, il n'existe pas de processus visité par une vague q avec $q \neq p$.

2.6 Mises en œuvre

Le schéma de contrôle abstrait présenté qui réalise une itération contrôlée par un processus P_α peut être mis en œuvre de diverses façons. Nous en détaillerons deux qui correspondent aux mises en œuvre "canoniques" et qui s'expriment sur les structures de contrôle que sont l'anneau et l'arborescence couvrante (le lecteur intéressé trouvera dans [HR88b] des algorithmes répartis de construction de telles structures). Il est fondamental de constater que la définition précédente n'est pas tributaire de ces mises en œuvre. (La démarche est ici la même que celle qui a prévalu à la définition des types de données abstraits [LZ74]). Pour chaque mise en œuvre, nous donnons le texte de chacune des 4 primitives.

2.6.1 L'anneau virtuel

Il est toujours possible de construire un anneau virtuel dans un réseau de processus [HR88a]: il s'agit d'une structure de communication contenant chaque processus une et une seule fois ; chaque processus P_i est alors pourvu de 2 informations: au niveau de la définition de l'anneau $succ_i$ désigne son successeur logique (ce n'est pas nécessairement un de ses voisins dans le réseau) et, au niveau de la mise en œuvre de l'anneau, un tableau R_i qui lui permet d'effectuer le routage adéquat : lorsqu'il reçoit une information en provenance de son voisin P_j, $R_i[j]$ lui indique vers lequel de ses voisins il faut la diriger.

L'anneau virtuel est une structure de communication qui va de pair avec le message de contrôle qu'est un *jeton* : celui-ci, en exemplaire unique, peut de façon évidente parcourir tous les processus depuis P_α avant d'y retourner. Le flot de contrôle *vague(diffusé,collecté)* est alors mis en œuvre par un unique message *jeton(diffusé,collecté)* et les 4 primitives deviennent:

- Pour P_α:

  lancer *vague(diffusé,∅)*: **envoyer** *jeton(diffusé, ∅)* **à** P_{succ_α}
  retour *vague(diffusé,collecté)*: **recevoir** *jeton(diffusé,collecté)*

- Pour P_i:

  visite *vague(diffusé,collecté)*: **recevoir** *jeton(diffusé,collecté)*
  faire-suivre *vague(diffusé,collecté)*:
  envoyer *jeton(diffusé, collecté)* **à** P_{succ_i}

2.6.2 L'arborescence

Il est toujours possible de construire une arborescence couvrante de racine P_α dans un réseau connexe de processus communiquant par canaux bidirectionnels.

Du point de vue d'une telle arborescence tout processus P_i est pourvu des 2 informations: $père_i$ et $fils_i$ qui désignent parmi les voisins de P_i respectivement le père et les fils de P_i dans cette structure; ($père_\alpha = nil$ par définition; si $fils_i = \emptyset$ alors P_i est une feuille). Le flot de contrôle *vague(diffusé,∅)* issu de P_α est propagé à l'aide de messages *aller(diffusé)* vers ses fils qui deviennent alors atteints par la vague et la propagent à leur tour vers leurs propres fils. Lorsque la vague atteint, à l'aide d'un message *aller*, une feuille P_k de l'arborescence couvrante elle est réfléchie par P_k qui la fait progresser vers

son père à l'aide d'un message *retour(collecté)*. Lorsqu'un processus a reçu un message *retour* de chacun de ses fils il rassemble les valeurs *collecté* et les envoie à son propre père à l'aide d'un message *retour*. Comme on le voit la vague descend le long des branches de P_α jusqu'aux feuilles (messages *aller(diffusé)*), s'y réfléchit et remonte jusqu'à la racine P_α, chaque processus P_i synchronisant la remontée du flot du contrôle qui le concerne (attente de messages *retour* de chacun de ses fils avant d'en envoyer à son père).

Les 4 primitives sont alors réalisées par les textes suivants (r_i est la contribution de P_i à la vague courante).

Pour P_α:

lancer *vague(diffusé,∅)*:
$\forall x \in fils_\alpha$: **envoyer** *aller(diffusé)* **à** P_x

retour *vague(diffusé,collecté)*:
$\forall\, x \in fils_\alpha$:
recevoir *retour* ($collecté_x$) **de** P_x;
collecté:= $\bigcup$ $collecté_x$

Pour $P_i (i \neq \alpha)$:

visite *vague(diffusé,collecté)*:
recevoir *aller(diffusé)*;
$\forall x \in fils_i$: **envoyer** *aller(diffusé)* **à** P_x

faire suivre *vague(diffusé,collecté)*:
si $fils_i \neq \emptyset$
alors
$\forall\, x \in fils_i$:
recevoir *retour*($collecté_x$) **de** P_x;
collecté := $\bigcup collecté_x$;
envoyer *retour*(*collecté* $\cup \{r_i\}$) **à** $P_{père_i}$
sinon
envoyer *retour*(r_i) **à** $P_{père_i}$
fsi

Remarque : Les valeurs collectées r_i sont accumulées dans un ensemble tel que, lorsque la vague est terminée, on a en P_α:

$$collecté = r_1 \cup r_2 \cup \ldots \cup r_n$$

Selon le calcul F que leur applique P_α, il peut être possible de distribuer ce calcul dans les divers processus, par exemple si F met en jeu des opérateurs associatifs [HR88b].

3 Illustration: un calcul des chemins de valeur minimale

3.1 Rappel: le problème et l'algorithme de Dijkstra

On considère un graphe orienté valué $G = (X, \Gamma, v)$ où v est une fonction $\Gamma \rightarrow R^+$ prolongée sur l'ensemble des chemins de G par addition. Etant donné un sommet x_α on cherche pour tout autre sommet x_i les valeurs de 2 attributs λ_i et $pred_i$ qui représentent respectivement la valeur minimale des chemins de x_α à x_i et le prédécesseur de x_i sur l'un des chemins de valeur minimale de x_α à x_i (par convention $\lambda_\alpha = 0$ et $pred_\alpha$ est non défini).

L'algorithme séquentiel proposé par Dijkstra est fondé sur une exploration de la descendance de x_α construisant une arborescence de chemins de valeurs minimales issus de x_α; l'hypothèse de positivité sur les valeurs des arcs permet de mener cette exploration de manière gloutonne, c'est-à-dire sans retour en arrière (un chemin de valeur minimum ne peut être remis en question). L'algorithme est itératif : chaque pas de l'itération permet de sélectionner un sommet dont les attributs deviennent définitifs, et de prolonger l'exploration aux successeurs de ce sommet. La description formelle de l'itération est alors la suivante:

Invariant : AE est une arborescence d'exploration, de racine x_α. Chaque sommet x se trouve dans l'un des trois états suivants:

dehors: le sommet x ne fait pas partie de AE; autrement dit, aucun chemin de x_α à x n'a encore été évalué,

atteint: le sommet x est une feuille de AE; ses attributs λ_x et $pred_x$ ont donc une valeur, égale respectivement à la valeur minimum des chemins de x_α à x évalués jusque là, et au père de x dans AE. La branche de x_α à x dans AE correspond à un chemin de G, reliant x_α à x, de valeur λ_x et d'avant dernier sommet $pred_x$. Ces valeurs ne sont pas nécessairement définitives, et l'exploration n'a pas encore été prolongée à partir de x,

calculé : le sommet x est dans AE et ses attributs $_x$ et $pred_x$ ont obtenu leurs valeurs définitives. De plus, l'exploration a été pro-

longée à partir de x : ses successeurs dans G sont tous dans l'état *atteint* ou *calculé*.

La sous-arborescence de AE engendrée par les sommets dans l'état *calculé* est l'arborescence des chemins de valeur minimum vers ces sommets, elle est stable; aucun pas ultérieur de l'itération ne pourra la remettre en cause.

Critère d'arrêt : il exprime que tous les descendants de x_α sont dans l'état *calculé* (les autres sommets étant dans l'état *dehors*, c'est à dire inaccessibles):

$$\{x | état_x = atteint\} = \emptyset$$

Règle de progression (calcul effectué lors d'un pas de l'itération) : il consiste à faire passer un sommet de l'état *atteint* à l'état *calculé*, puis à prolonger l'exploration en examinant l'état de tous les successeurs de ce sommet:

soit *nouv* le sommet d'attribut λ minimum
parmi tous les sommets dans l'état *atteint*;
$état_{nouv} := calculé$;
$\forall y \in \Gamma(nouv)$:
cas $état_y = dehors \longrightarrow$
$état_y := atteint; \lambda_y := \lambda_{nouv} + v(nouv, y); pred_y := nouv$
$état_y = atteint \longrightarrow$
si $\lambda_y > \lambda_{nouv} + v(nouv, y)$
alors $\lambda_y := \lambda_{nouv} + v(nouv, y); pred_y := nouv$
fsi
$état_y = calculé \longrightarrow skip$
fcas

Situation initiale : $état_\alpha = calculé, \lambda_\alpha = 0, pred_\alpha$ non défini;
$\forall y \in \Gamma(x_\alpha)$: $état_y = atteint, \lambda_y = v(x_\alpha, y), pred_y = x_\alpha$;
pour les autres sommets x: $état_x = dehors$.

3.2 Répartition en réseau de processus

La formulation répartie du problème résolu par l'algorithme de Dijkstra définit un réseau de processus dont la structure épouse celle du graphe

$G = (X, \Gamma, v)$: un processus et un canal (unidirectionnel) correspondent respectivement à un sommet et à un arc. [1]

Initialement, chaque processus P_i est doté d'une certaine connaissance locale (il ne connait pas la structure globale du réseau, ni même le nombre de processus):

- l'ensemble des identités de ses successeurs: $succ_i$
- la valeur associée à chacun de ses canaux sortants (P_i, P_j): $valc_i(j)$.

Une fois construite, l'arborescence de racine P_α, doit pouvoir être exploitée, de façon distribuée. A la fin de la construction chaque processus P_i doit donc posséder les informations suivantes:

i) sur le routage descendant: pour chacun de ses descendants P_j dans l'arborescence, vers lequel de ses fils P_i doit-il faire suivre les messages destinés à P_j?

ii) sur le routage ascendant: il s'agit de l'attribut $pred_i$ défini précédemment

iii) la valeur de l'attribut λ_i

iv) éventuellement, en ce qui concerne P_α , l'ensemble des attributs λ_i.

Ces informations seront mémorisées, sur chacun des processus, dans les variables locales suivantes:

- pour tous les processus P_i:

$succ\_opt_i$: **tableau de identité de processus**, *avec*

$$succ\_opt_i[j] = \begin{cases} k \text{ si } P_j \text{ est descendant de } P_i \\ \text{dans l'arborescence construite} \\ \quad \text{et } P_k \text{ est le successeur de } P_i \text{ sur la branche} \\ \text{allant de } P_i \text{ à } P_j \\ \\ \text{non défini (nil) si } P_j \text{ n'est pas descendant de } P_i \\ \quad \text{dans l'arborescence} \end{cases}$$

[1] Nous considérons le cas, plus général, où les canaux sont logiquement unidirectionnels et valués; nous distinguons donc le couple (P_i, P_j) – de valeur v_{ij} – du couple (P_j, P_i) – de valeur v_{ji}. Toutefois, nous admettrons qu'il est possible de faire circuler des messages de contrôle dans les deux sens: les canaux sont physiquement bi-directionnels.

- pour l'initiateur P_α:

$$\begin{aligned} dmin_\alpha &: \textbf{tableau de réel, avec} \\ dmin_\alpha[i] &= \text{valeur minimale des chemins de } P_\alpha \text{ à } P_i \\ &= \text{valeur de la branche allant de } P_\alpha \text{ à } P_i \end{aligned}$$

(si P_i n'est pas atteignable depuis P_α alors $dmin_\alpha[i]$=**nil**)

- pour tout processus $P_i, i \neq \alpha$:

λ_i: **réel**

$pred_i$: **identité_de_processus**

3.3 Répartition de l'itération

Initialement, P_α est dans l'état *calculé* et place ses voisins sont dans l'état *atteint*. Un pas d'itération correspond à une vague lancée par P_α dont le rôle est double:

- *diffuser* l'identité du processus sélectionné à cette étape (celui dont l'état passe de *atteint* à *calculé*) et, lorsque ce dernier est visité par la vague, "examiner ses successeurs",
- *collecter* les attributs λ et les identités des processus dont l'état vaut *atteint* à la *fin* de la vague. Au retour de la vague, P_α sera ainsi en mesure de déterminer si l'algorithme est terminé (lorsqu'il n'y a plus aucun processus dans l'état *atteint*), ou le processus à sélectionner pour la prochaine étape, dont l'identité sera diffusée par la vague suivante.

3.4 Invariant, critère d'arrêt et règle de progression

La conception de l'algorithme réparti se ramène à la définition de ces 3 éléments. L'invariant décrit la relation globale vérifiée au retour de chaque vague, la règle de progression décrit les actions effectuées par les processus lorsqu'ils sont visités par la vague.

La vague courante *vague(diffusé,collecté)* véhicule les informations suivantes:

diffusé = *nouv*: identité du site sélectionné

collecté = (imin, vmin): identité et attribut du site *atteint* ou devenant *atteint*, d'attribut λ minimum, visité par la vague courante (c'est le prochain site à sélectionner).

Enfin, tout processus P_i qui n'est pas encore dans l'état *calculé* maintient, dans une variable $encom_i$, l'ensemble des identités de ses successeurs qui ne sont pas encore dans l'état *calculé*. Cette information sera utilisée par le processus P_i lorsqu'il sera sélectionné pour passer dans l'état *calculé*, lors d'une vague ultérieure: à ce moment là, il aura à prolonger l'exploration vers ceux de ses successeurs qui ne sont pas dans l'état *calculé*, donc seulement vers les processus dont l'identité est mémorisée dans $encom_i$; une économie de messages est ainsi réalisée, car la mise à jour de $encom_i$ est immédiate à partir de l'information diffusée par chaque vague (aucun message supplémentaire n'est requis).

3.4.1 Invariant

- sur tout processus P_i:

  % $état_i$ = état du processus (*dehors, atteint,* ou *calculé*) %
  début $état_i = dehors \Rightarrow$
  $encom_i$ = ensemble des successeurs non calculés;
  $\wedge$ $état_i = atteint \Rightarrow$
  $encom_i$ = ensemble des successeurs non calculés;
  λ_i, $pred_i$: attributs (non définitifs)
  $état_{pred_i} = calculé$
  $\wedge$ $état_i = calculé \Rightarrow$
  λ_i, $pred_i$: attributs définitifs et corrects
  $état_{pred_i} = calculé$;
  $succ\_opt_i[j]$ est défini et correct pour tout descendant P_j dans l'arborescence, tel que $état_j = calculé$
  fin

- pour le processus P_α:

  nouv, *vnouv* = identité et attribut λ du processus sélectionné pour la prochaine vague (résultat de la collecte précédente);
  nouv = **nil ssi** il n'y a plus de processus atteint

$$\forall i \in X : dmin[i] = \begin{cases} \text{valeur minimale des chemins de } P_\alpha \text{ à } P_i \\ \quad \text{si } état_i = calculé, \\ \text{nil sinon} \end{cases}$$

3.4.2 Condition d'arrêt

(vérifiée par P_α) $collecté.imin = \text{nil}$

3.4.3 Progression

La contribution de chaque processus P_i à l'information *collecté* se fait lorque P_i est visité par la vague:

s'il est *dehors* au moment de la visite, il ne contribue pas à la collecte; par contre, il apprend que *nouv* est le processus devenant *calculé* à cette étape, d'où la mise à jour: $encom_i := encom_i - \{nouv\}$.

s'il est *atteint* au moment de la visite, deux situations sont à distinguer:

- soit $P_i \neq nouv$; dans ce cas, P_i fournit sa contribution (i, λ_i) et met à jour sa variable $encom_i$ comme ci-dessus,
- soit $P_i = nouv$; dans ce cas, sa contribution doit être remplacée par celle de ses successeurs qui ne sont pas dans l'état *calculé* (mémorisés localement dans $encom_i$). En effet, chacun de ceux-ci a pu être visité par la vague *avant* P_i, et donc avoir contribué à la collecte de manière caduque (si par exemple son état ou ses attributs doivent être modifiés par *nouv*); d'autre part, *nouv*, passant dans l'état *calculé*, ne participe plus à la collecte. Il en résulte que le processus *nouv* se charge de la contribution des processus de $encom_{nouv}$; il en a les moyens: en effet, l'attribut λ_j d'un tel processus P_j sera, à la fin de la vague, inférieur ou égal à la valeur $\lambda_{nouv} + valc_{nouv}(j)$, et cette quantité est connue sur le processus *nouv*. La contribution fournie par ce dernier est donc:

$$(jmin, \lambda min) \text{telle que :}$$

$$\lambda min = \lambda_{nouv} + \min_{j \in encom_{nouv}} (valc_{nouv}(j)) \text{ atteint pour } jmin.$$

Enfin, P_i envoie vers chacun des processus $P_j \in encom_i$ un message *modifier(v)*, avec $v = \lambda_{nouv} + valc_{nouv}(j)$ ce qui permet la

mise à jour de l'état et/ou des attributs de ces processus (mise en œuvre de l'action "*examiner les successeurs non calculés du processus sélectionné* " du §3.3). Cette mise à jour devant prendre effet avant le passage de la prochaine vague, une synchronisation est nécessaire; cette synchronisation ne peut se faire sur la *réception* des messages *modifier* puisque les processus concernés ne savent pas, a priori, s'ils font ou non partie de l'ensemble $encom_{nouv}$; c'est donc au niveau du processus *nouv* lui- même qu'elle se fera; une technique simple est celle des *acquittements,* permettant à un processus émetteur P d'apprendre, lorsque les délais de transmission sont imprévisibles, que le récepteur a bien reçu le message; ainsi, *nouv* devra attendre d'avoir reçu tous les acquittements, pour faire suivre la vague (bien sûr, selon les hypothèses de communication, d'autre techniques peuvent être discutées). Nous avons donc ici un exemple de vague dont la progression est soumise localement à la satisfaction d'une condition de synchronisation, due à l'application et non pas à la mise en œuvre de la vague; un tel schéma se retrouvera dans d'autres applications ([Awe85]).

D'autre part, les attributs de $P_i(i = nouv)$ deviennent définitifs: P_i, ainsi que tous ses ancêtres dans l'arborescence, sont dans l'état *calculé* et donc la branche aboutissant à P_i est stable; tous les processus P_k situés sur cette branche peuvent alors mettre à jour leur table $succ\_opt_k[nouv]$: pour les en informer, *nouv* fait remonter, le long de cette branche (grâce aux attributs *pred*),un message du type *routage(nouv)*. Il n'est pas nécessaire de synchroniser, en P_α, le retour de la vague avec la réception du message *routage* relatif à cette vague (ce dernier peut cheminer sur une structure éventuellement différente de la structure de parcours de la vague); il suffit de renforcer la condition de terminaison de l'algorithme, testée par P_α: tous les messages *routage* portant les identités diffusées lors des vagues successives doivent être reçus.

- S'il est *calculé* au moment de la visite, il se contente de faire suivre la vague (il ne participe plus au calcul).

3.4.4 Initialisation

Le comportement de la première vague est plus simple, puisqu'initialement, $nouv = \alpha$: cette première vague se résume donc à l'envoi, par P_α, de messages

modifier vers chacun de ses successeurs, et de l'attente des acquittements correspondants.

La correction partielle de l'algorithme découle des trois résultats suivants:

1. L'invariant est satisfait à l'issue de l'initialisation,
2. La progression maintient l'invariant,
3. La conjonction de l'invariant et de la condition d'arrêt implique le résultat (pour ce dernier point, on s'appuie sur la démonstration déjà connue en contexte centralisé séquentiel).

La terminaison est garantie dès lors que les délais de transmission des messages sont finis.

3.5 L'algorithme réparti

Le texte de l'algorithme est donné figure 1.

3.6 Remarques

3.6.1 Mises en œuvre

Comme cela a été montré au §2.6, le train de vagues sur lequel s'appuie cet algorithme peut être mis en œuvre à l'aide de différents schémas de contrôle et structures de parcours associées (séquentiel/anneau, parallèle/arborescence); dans chacun des cas, il suffit de traduire les primitives d'accès au schéma de contrôle abstrait que constitue la vague.

3.6.2 Complexité en nombre de messages

A chaque vague, un nouveau processus passe dans l'état *calculé*, donc il y a au plus n vagues, où n est le nombre de processus. Au cours de la vague $n°k$, si l'on désigne par $nouv_k$ le processus sélectionné et par $d^+(x)$ le nombre de successeurs de x, il y a:

au plus $\min(d^+(nouv_k), n-k)$ messages *modifier* et *acq*

au plus k-1 messages *routage*

Comportement de l'initiateur P_α

$état_\alpha$:= *calculé*; λ_α:= 0; $pred_\alpha$:= α; $dmin_\alpha[\alpha]$:= 0;
$\forall j \in succ_\alpha$: **envoyer** *modifier*($valc_\alpha(j)$) **à** P_j;
$nbatt_\alpha := card(succ_\alpha)$;
attendre($nbatt_\alpha$ = 0, réception de messages *acq*);
soit (*nouv,vnouv*) tel que $vnouv = \min_{j \in succ_\alpha} valc_\alpha(j)$,
atteint pour j=*nouv*;
tant que *nouv* $\neq$ *nil* **faire**
dmin$_\alpha$[*nouv*] := *vnouv*;
lancer *vague*(*nouv*, **nil**, /);
retour *vague*(*nouv, imin, vmin*);
nouv := *imin*; *vnouv*:= *vmin*
fintantque;
% ici, P_α arrête de lancer des vagues
car il n'y a plus de sommet atteint %
attendre ($\forall k$ tel que $dmin[k] \neq nil$: $succ\_opt_\alpha[k] \neq nil$,
réception de messages *routage*);
% algorithme terminé %

Figure 1 : a

Comportement d'un processus P_i $(i \neq \alpha)$

lors de visite *vague(nouv, imin, vmin)*
  cas $état_i$ =*dehors* $\longrightarrow$
    $encom_i := encom_i - \{nouv\}$;
  $état_i$ = *atteint* $\longrightarrow$
    si i =*nouv*
      alors % P_i est le processus sélectionné à cette étape %
        $état_i$:= *calculé*;
        envoyer *routage(i)* **à** P_{pred_i};
        $\forall j \in encom_i$: **envoyer** $modifier(\lambda_i + valc_i(j))$;
        $nbatt_i := card(encom_i)$;
        attendre($nbatt_i = 0$) % réception de messages *acq*%;
        soit (*jmin*, λmin) **tel que** $\lambda min = \min_{j \in encom_i} valc_i(j)$,
                                atteint pour j = *jmin*;
                                % *jmin* = *nil* si $encom_i = \emptyset$ %
        si *jmin* $\neq$ *nil*
          alors si $\lambda min < vmin$ **ou** *imin* = *nil*
            alors *vmin*:= λmin; *imin*:= *jmin*
          fsi
        fsi
      sinon $encom_i := encom_i - \{nouv\}$;
        si $\lambda_i < vmin$ **ou** *imin* = *nil*
          alors *vmin*:= λ_i; *imin*:= i
        fsi
    fsi;
  $état_i$ = *calculé* $\longrightarrow$ *skip*
fcas;
faire_suivre *vague(nouv, imin, vmin)*

Figure 1 : b

Comportement d'un processus quelconque P_i vis-à-vis des messages

lors de réception de *modifier(v)* **depuis** P_j % P_j=*nouv* %
    cas $état_i$ = *dehors* $\longrightarrow$
        $état_i$:= *atteint*; λ_i:= v; $pred_i$:= j;
    $état_i$ = *atteint* $\longrightarrow$
        si $\lambda_i > v$ **alors** λ_i:= v; $pred_i$:= j **fsi**
    fcas;
    envoyer *acq* **à** P_j

lors de réception de *acq* **depuis** P_j % P_i = *nouv* %
    $nbatt_i := nbatt_i - 1$

lors de réception de *routage(k)* **depuis** P_j **faire** % $état_i$ = *calculé*%
    $succ\_opt_i[k]$:= j;
    si $pred_i \neq i$ % $P_i \neq P_\alpha$ %
      alors envoyer *routage(k)* **à** P_{pred_i}
    fsi

Figure 1 : c

Si une vague nécessite p messages (sur l'anneau ou l'arborescence, on a $p = O(n)$), on obtient, au pire:

$(n-1)p$ messages de gestion des vagues

$\sum_{k=1}^{n} \min(d^+(nouv_k), n-k) \leq e$ messages *modifier* et *acq*

$\sum_{k=1}^{n}(k-1) = \frac{n(n-1)}{2}$ messages *routage*

où e est le nombre d'arcs du réseau, d'où une complexité en nombre de messages en $O(n^2)$.

3.6.3 Amélioration sur une arborescence

Le problème posé présente une particularité intéressante du point de vue de la mise en œuvre des vagues; celle-ci peut, vu le calcul effectué, être supportée par une arborescence partielle : l'arborescence des chemins de valeurs minimales déjà construite! Le schéma n'est plus alors abstrait pusiqu'il y a une étroite imbrication entre le calcul et la mise en œuvre des objets utilisés. Il est dès lors intéressant de constater que si l'on considère une telle démarche et si les valeurs associées aux arêtes sont toutes égales (à 1) on retrouve l'algorithme particulier à ce cas proposé par [ZC87].

4 Conclusion

Selon que l'évaluation du critère de terminaison est distribuée dans l'ensemble des processus ou centralisée dans un seul d'entre eux, deux formes d'itérations réparties peuvent être distinguées. Nous avons étudié la seconde dans cet article; elle s'appuie sur la définition d'un schéma de contrôle abstrait: la *vague*. Ce schéma est défini par 4 primitives indépendamment d'une mise en œuvre particulière; la démarche adoptée se veut donc un prolongement de celle qui, dans le domaine séquentiel, a conduit aux concepts de structures de contrôle et de données abstraites.

L'illustration proposée : calcul réparti de chemins de valeurs minimales, est intéressante à double titre. D'une part, un nouvel algorithme distribué a été obtenu, performant quant à sa complexité en nombre de messages (grâce à l'exploitation, dans le contexte distribué, de l'hypothèse de positivité des valeurs des arcs), et offrant une solution complète au problème posé (notamment par les informations de routage descendant). D'autre part, par sa conception méthodique: les éléments fondamentaux de l'itération (répartie)

ont d'abord été précisés: invariant, critère d'arrêt, règles de progression, puis les mécanismes de synchronisation (et les messages de contrôle afférents) nécessaires à l'établissement des règles de progression ont été définis. Une telle démarche nous paraît prometteuse car, comme nous l'avons indiqué, non seulement elle permet de définir de nouveaux (et performants) algorithmes mais de plus elle permet d'expliquer les principes sous-jacents à des algorithmes existants et par conséquent, en facilite l'étude et la comparaison.

Bibliographie

[Awe85] B. AWERBUCH. A new distributed depth first search algorithm. *Inf. Proc. Letters*, 20:147–150, 1985.

[BKR87] J.C. BERMOND, J.C. KONIG, and M. RAYNAL. General and efficient decentralised consensus protocols. In *Proc. 2d Int. Worhshop a Distributed Algorithms*, Amsterdam, July 1987. LNCS 312, Springer-Verlag.

[CM82] K.M. CHANDY K.M., J. MISRA. Distributed Computation on Graphs : Shortest paths Algorithms. *Comm. ACM*, 25 (11):833-837, Nov. 1982.

[Chan82] E.J.H. CHANG. Echo algorithms : depth parallel operations on general graphs. *IEEE Trans. on SE*, SE8(4):391–401, July 1982.

[Che83] T. CHEUNG. Graph travernal techniques and the maximun flow problem in distributed computation. *IEEE Trans. on SE*, SE 9(4):504–512, 1983.

[Dij59] E.W.D. DIJKSTRA. A note on Two Problems in Connexion with Graphs. *Numerische Matematik*, 1:269-271, 1959.

[Gaf86] E. GAFNI. Perspectives on distributed network protocols: a case for building blocks. *Proc. IEEE Military Communications Conf.*, Monterey, oct. 1986.

[Gal82] R. G. GALLAGER. *Distributed Minimum Hop Algorithms.* Technical Report LIDS- p 1175, M.I.T., Jan. 1982.

[HR88a] J.M HELARY, M. RAYNAL. Virtual Ring Construction in Parallel Distributed Systems. In *Proc. IFIP WG 10.3 Working*

Conf. on Parallel Processing, Pisa, Italy, April 1988. to appear : North Holland.

[HR88b] J.M. HELARY, M. RAYNAL. *Synchronisation et contrôle des Systèmes et Programmes Répartis.* Eyrolles, Paris, septembre 1988, 195p.

[HR88c] J. M. HELARY, M. RAYNAL. Les parcours distribués de réseaux: un outil pour la conception de protocoles. *Actes du Colloque CFIP'88*, Bordeaux, 20-22 septembre 1988. à paraître, Eyrolles ed.

[Hoa74] C. A. R. HOARE. Monitors : an operating system structuring concept. *Comm. ACM*, 17(10):549–557, Oct. 1974.

[Kon87] J. C. KONIG. *Les réseaux d'interconnexion et les algorithmes distribués.* thèse, Université Paris-Sud-Orsay, Avril 1987.

[LZ74] B. LISKOV, S.N. ZILLES. Programming with Abstract Data Types. *Proc. ACM Sigplan Conf. on Very High Level Languages*, (April 1974), pp 50-59.

[Seg83] A. SEGALL. Distributed network protocols. *IEEE Trans. on Inf. Theory*, IT 29(1):23–35, jan. 1983.

[Sch85] F.P. SCHNEIDER Paradigms for Distributed Programs. In *Distributed Systems*, LNCS 190: 431-480, Springer-Verlag Ed., (1985).

[Tel87] G. TEL. Directed network protocols. *Proc. 2d. Int. Workshop on Distributed Algorithms*, LNCS 312, Springer-Verlag, 1988.

[Tel88] G. TEL. *Total Algorithms.* RUU-CS-88-16, University of Utrecht, (April 1988), 23p.

[WSL77] W.A. WULF, M. SHAW, R.L. LONDON. Abstraction and verification in ALPHARD : defining and specifying iteration and generators. *Comm. ACM*, 20(8):553-563, Aug. 1977.

[ZC87] Y. ZHU and T. Y. CHEUNG. A new distributed breadth-first search algorithm. *Inf. Processing Letters*, 25:239–333, 1987.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique